CrackMe 6 – TTO

solution par mastermatt29

1.Introduction

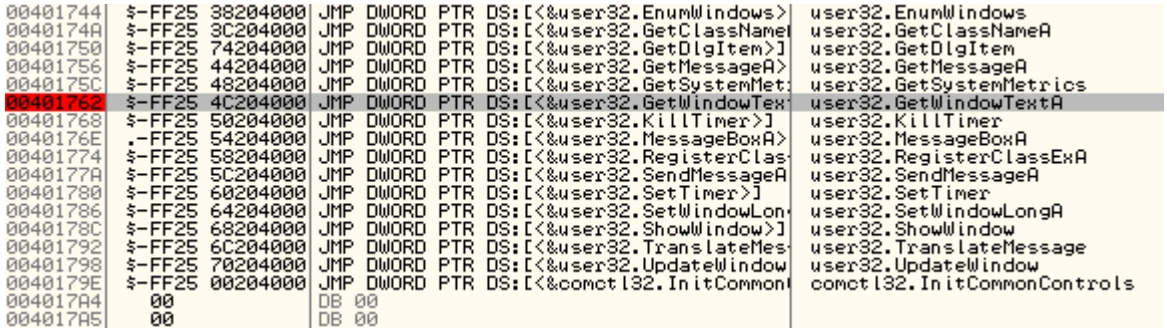
Je propose ici une solution au crackme 6 de Thierry The One. Ce tutorial fait partie d'un projet initié par Gamera pour la NAS (powaaaa) et regroupant les 8 solutions des 8 crackme de TTO.

Dans la suite de ce tutorial, **toute valeur sera systematiquement exprimée en héxadécimal** sauf indication contraire.

2.Anti-Debug

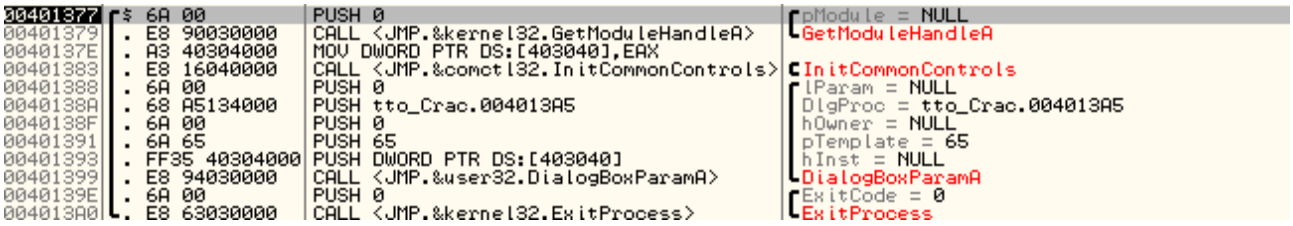
a) Généralités

Allez on est parti ! On lance le crackme et on observe...3 edits, un pour le name, et 2 pour des serials.
Le crackme n'est pas packé, on peut donc le charger directement dans OllyDbg. Direction le bas du listing là ou sont tous les JMP DWORD PTR... On pose un breakpoint sur GetWindowTextA (l'une des fonctions les plus courantes pour récupérer du texte) et on lance avec F9.



La ptite surprise c'est qu'au bout d'un temps très court Olly est tout simplement fermé ! TTO avait prévenu, il y a des anti-debug !

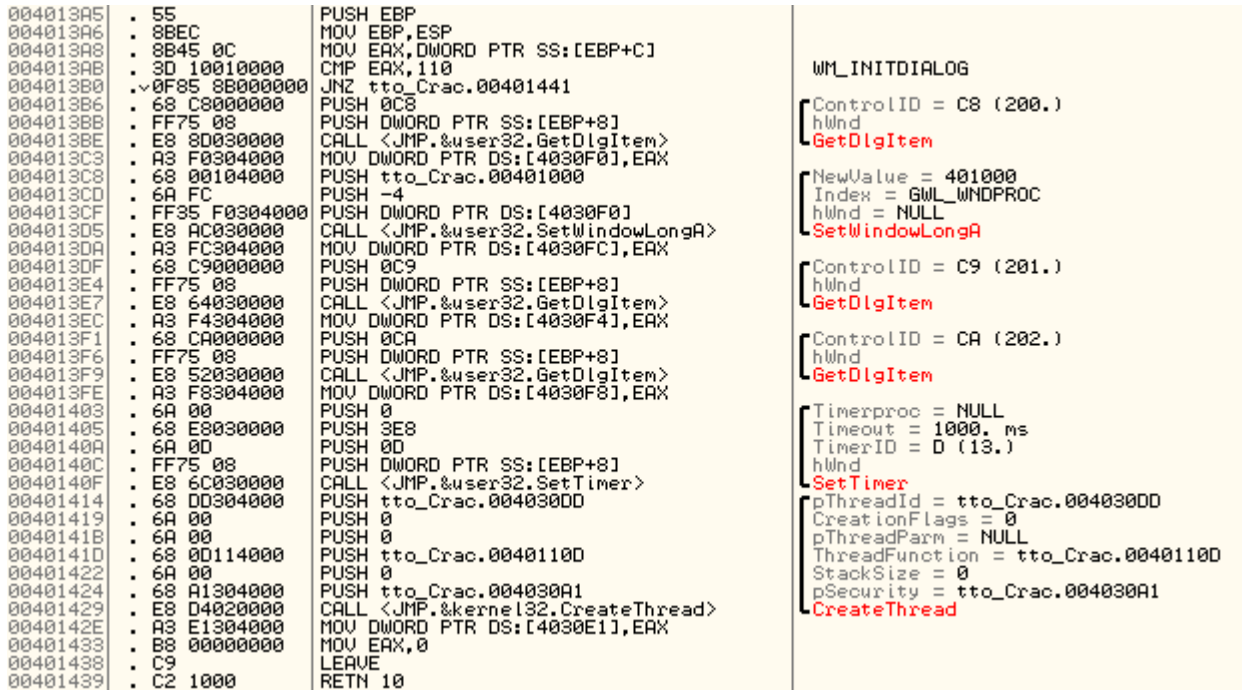
Bon ben on va tracer au fur et à mesure alors ! C'est ptête pas très réjouissant mais au moins on verra bien comment fonctionne le code. Reprenons depuis le début :



Première chose que l'on remarque l'appel à DialogBoxParamA qui indique que la fonction Callback du programme est située en 4013A5 (paramètre DlgProc). Il faut aussi savoir que cette fonction envoie un message WM_INITDIALOG à la Callback sitot son travail terminé. Il est donc logique d'aller examiner la Callback et voir ce qu'elle fait lorsqu'elle traite un message WM_INITDIALOG.

Note : la fonction Callback d'un programme est chargée de traiter des messages fournis par Windows (par exemple si une touche à

été appuyée, si on demande la fermeture, si on redimmensionne la fenêtre...). Chaque fenêtre peut posséder sa fonction Callback.



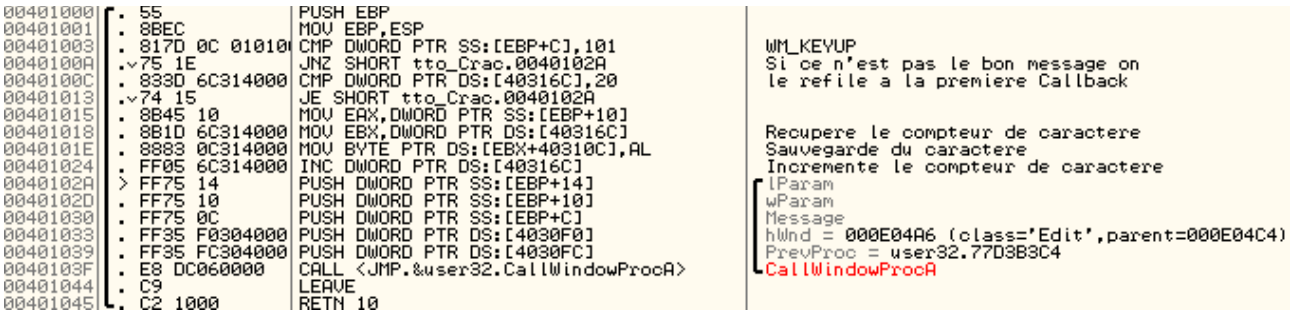
```
004013A5  . 55              PUSH EBP
004013A6  . 8BEC            MOV EBP,ESP
004013A8  . 8B45 0C         MOV EAX,DWORD PTR SS:[EBP+C]
004013AB  . 3D 10010000     CMP EAX,110
004013B0  .v0F85 8B000000   JNZ tto_Crac.00401441
004013B6  . 68 C8000000     PUSH 0C8                          WM_INITDIALOG
004013BB  . FF75 08         PUSH DWORD PTR SS:[EBP+8]         ControlID = C8 (200.)
004013BE  . E8 8D030000     CALL <JMP.&user32.GetDlgItem>     hWnd
004013C3  . A3 F0304000     MOV DWORD PTR DS:[4030F0],EAX     GetDlgItem
004013C8  . 68 00104000     PUSH tto_Crac.00401000            NewValue = 401000
004013CD  . 6A FC           PUSH -4                           Index = GWL_WNDPROC
004013CF  . FF35 F0304000   PUSH DWORD PTR DS:[4030F0]        hWnd = NULL
004013D5  . E8 AC030000     CALL <JMP.&user32.SetWindowLongA> SetWindowLongA
004013DA  . A3 FC304000     MOV DWORD PTR DS:[4030FC],EAX
004013DF  . 68 C9000000     PUSH 0C9                          ControlID = C9 (201.)
004013E4  . FF75 08         PUSH DWORD PTR SS:[EBP+8]         hWnd
004013E7  . E8 64030000     CALL <JMP.&user32.GetDlgItem>     GetDlgItem
004013EC  . A3 F4304000     MOV DWORD PTR DS:[4030F4],EAX
004013F1  . 68 CA000000     PUSH 0CA                          ControlID = CA (202.)
004013F6  . FF75 08         PUSH DWORD PTR SS:[EBP+8]         hWnd
004013F9  . E8 52030000     CALL <JMP.&user32.GetDlgItem>     GetDlgItem
004013FE  . A3 F8304000     MOV DWORD PTR DS:[4030F8],EAX
00401403  . 6A 00           PUSH 0
00401405  . 68 E8030000     PUSH 3E8                          Timerproc = NULL
0040140A  . 6A 0D           PUSH 0D                           Timeout = 1000. ms
0040140C  . FF75 08         PUSH DWORD PTR SS:[EBP+8]         TimerID = D (13.)
0040140F  . E8 6C030000     CALL <JMP.&user32.SetTimer>       hWnd
00401414  . 68 DD304000     PUSH tto_Crac.004030DD            SetTimer
00401419  . 6A 00           PUSH 0                            pThreadId = tto_Crac.004030DD
0040141B  . 6A 00           PUSH 0                            CreationFlags = 0
0040141D  . 68 0D114000     PUSH tto_Crac.0040110D            pThreadParm = NULL
00401422  . 6A 00           PUSH 0                            ThreadFunction = tto_Crac.0040110D
00401424  . 68 A1304000     PUSH tto_Crac.004030A1            StackSize = 0
00401429  . E8 D4020000     CALL <JMP.&kernel32.CreateThread> pSecurity = tto_Crac.004030A1
0040142E  . A3 E1304000     MOV DWORD PTR DS:[4030E1],EAX     CreateThread
00401433  . B8 00000000     MOV EAX,0
00401438  . C9              LEAVE
00401439  . C2 1000         RETN 10
```

Chaque message possède une valeur particulière unique qui permet de le reconnaître et de faire des actions en conséquence. La valeur de WM_INITDIALOG est 110, le traitement approprié commence donc en 4013B6.

Regardons un peu. Les choses intéressantes sont les appels à SetWindowLongA, SetTimer et CreateThread. Nous allons les étudier successivement.

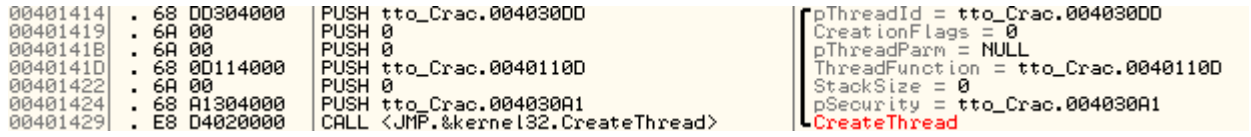
b) SetWindowLongA

Passons les appels à GetDlgItem qui n'ont guère d'intérêt et arrêtons nous sur l'appel à SetWindowLongA. Cette fonction sert à changer les paramètres d'une fenêtre, suivant le paramètre Index. Ici il s'agit de GWL_WNDPROC qui sert à modifier l'adresse de la Callback. Dans notre cas la nouvelle adresse est 401000 et contient du code intéressant :



Cette fonction se charge d'intercepter uniquement les message WM_KEYUP qui indiquent qu'une touche vient d'être relachée, tout autre message étant renvoyé vers la première Callback. Une fois le message intercepté, on stocke le caractère de la touche relevée en 40310C et on incrémente le nombre de caractères traités en 40316C. Cette manière de traiter les caractères fait que malheureusement même un appui sur Shift est enregistré. Nous verrons aussi plus tard qu'il s'agit spécifiquement du traitement du name.

c) CreateThread

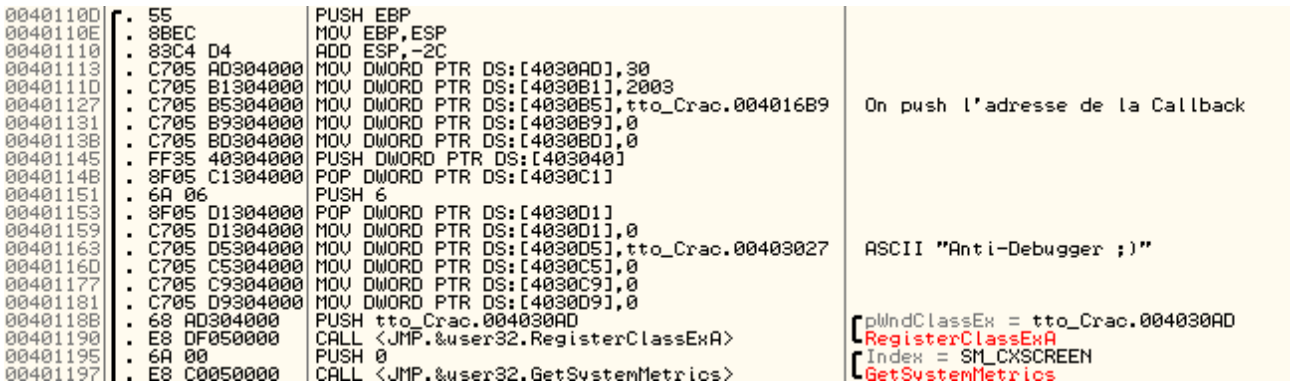


Il y a création d'un thread donc l'adresse de début est 40110D (paramètre ThreadFunction).

Note : un thread est une sorte de petit processus mais qui est lui interne au programme et partage son espace mémoire.

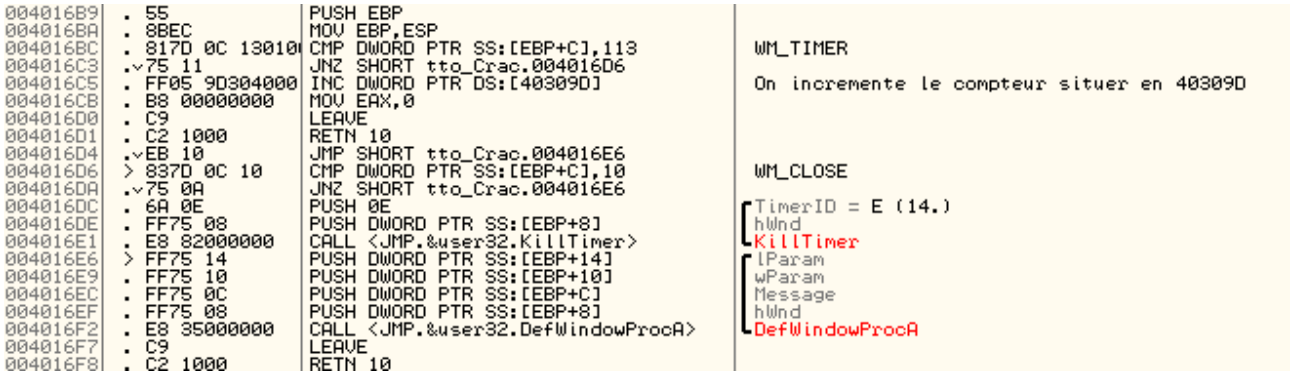
On tombe ici sur un début de programme courant : RegisterClassExA, CreateWindowExA, ShowWindow, UpdateWindow...et SetTimer ! C'est ce dernier qui va capter notre attention ! La fenetre associée est celle qui vient d'etre créée par CreateWindowExA et dont nous devons trouver la Callback (qui recevra donc les messages WM_TIMER).

Reflechissons....
On sait que RegisterClassExA prend en paramètre un pointeur sur une structure WNDCLASSEX qui est dans notre cas 4030AD. L'adresse de la Callback est le 3e DWORD de la structure WNDCLASSEX donc :



4016B9 est bien le 3e DWORD de la structure ! Si c'est pas formidable ca de trouver l'adresse comme ca en clair sans avoir besoin de debugger !

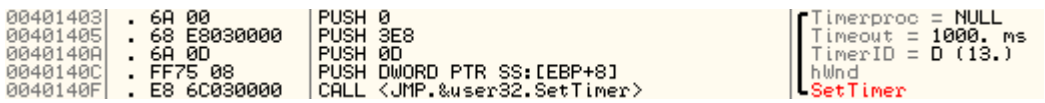
N'oublions pas que nous avons un petit SetTimer sur le feu. Faudrait ptete aller voir en 4016B9 maintenant !



La fonction de traitement des messages n'est pas bien grande. Si on a le message WM_TIMER alors on incrémente le compteur situé en 40309D sinon si c'est un WM_CLOSE on tue le timer.

d) SetTimer

Arrêtons nous maintenant sur le SetTimer (euh rien à voir avec le précédent, c'est celui encore avant).



On sait que le timer va envoyer des messages WM_TIMER (de valeur 113) à la Callback toute les secondes. On va donc suivre la liste des messages traités.

Le premier est WM_INITDIALOG (cf 2a). Ce n'est pas ca donc on continue en 401441

```
00401441| > 3D 11010000  |CMP EAX,111                          | WM_COMMAND
00401446| .v0F85 8E010000|JNZ tto_Crac.004015DA
```

Ici c'est WM_COMMAND (valeur 111), on continue en 4015DA.

```
004015DA| > 83F8 10      |CMP EAX,10                           | WM_CLOSE
004015DD| .v75 24        |JNZ SHORT tto_Crac.00401603
```

Toujours pas ! Allons en 401603.

```
00401603| > 3D 13010000  |CMP EAX,113                          | WM_TIMER
00401608| .v0F85 99000000|JNZ tto_Crac.004016A7                |
0040160E| . E8 D5FAFFFF  |CALL tto_Crac.004010E8               | <= 1er appel
00401613| . 50           |PUSH EAX                             | lParam
00401614| . 68 48104000  |PUSH tto_Crac.00401048               | Callback = tto_Crac.00401048
00401619| . E8 26010000  |CALL <JMP.&user32.EnumWindows>       | EnumWindows
0040161E| . E8 C5FAFFFF  |CALL tto_Crac.004010E8               | <= 2e appel
00401623| . 58           |POP EAX                              |
00401624| . 5B           |POP EBX                              |
00401625| . FF35 9D304000|PUSH DWORD PTR DS:[40309D]           | 40309D contient une valeur etudiee + tard
0040162B| . 3BC3         |CMP EAX,EBX                          |
0040162D| .v75 29        |JNZ SHORT tto_Crac.00401658          | Saut BadBoy
0040162F| . 3D B0000000  |CMP EAX,0B0                          |
00401634| .v75 22        |JNZ SHORT tto_Crac.00401658          | Saut BadBoy
00401636| . 25 FF000000  |AND EAX,0FF                          |
0040163B| . 3D B0000000  |CMP EAX,0B0                          |
00401640| .v75 16        |JNZ SHORT tto_Crac.00401658          | Saut BadBoy
00401642| . 5B           |POP EBX                              |
00401643| . A1 9D304000  |MOV EAX,DWORD PTR DS:[40309D]        |
00401648| . 2BC3         |SUB EAX,EBX                          |
0040164A| . 83F8 01      |CMP EAX,1                            | Entre le debut de la routine et ici, 40309D
0040164D| .v7F 09        |JG SHORT tto_Crac.00401658           | ne doit pas avoir augmenter de plus que 1
0040164F| . B8 00000000  |MOV EAX,0                            |
00401654| . C9           |LEAVE                                |
00401655| . C2 1000      |RETN 10                              |
00401658| > 6A 00        |PUSH 0                               | lParam = 0
0040165A| . 6A 00        |PUSH 0                               | wParam = 0
0040165C| . 6A 10        |PUSH 10                              | Message = WM_CLOSE
0040165E| . FF75 08      |PUSH DWORD PTR SS:[EBP+8]            | hWnd
00401661| . E8 14010000  |CALL <JMP.&user32.SendMessageA>      | SendMessageA
00401666| . B8 00000000  |MOV EAX,0                            |
0040166B| . C9           |LEAVE                                |
0040166C| . C2 1000      |RETN 10                              |
```

Ca y est ! On est tombé sur la routine qui traite les messages WM_TIMER.

Juste comme ca : vous remarquez peut-être que la valeur située entre 40309D ne doit pas avoir augmenté de plus de 1 entre le début et la fin de la routine. Nous verrons un peu plus loin de quoi il s'agit.

Ca commence avec un appel à 401048.

```
004010E8|$ 8D05 7B304000|LEA EAX,DWORD PTR DS:[40307B]       |
004010EE|. 48           |DEC EAX                              |
004010EF|. BB FFFFFFFF  |MOV EBX,-1                           |
004010F4|. B9 00000000  |MOV ECX,0                            |
004010F9|> 40           |INC EAX                              |
004010FA|. 43           |INC EBX                              |
004010FB|. 83FB 1E      |CMP EBX,1E                           |
004010FE|.v74 04        |JE SHORT tto_Crac.00401104           |
00401100|. 0208         |ADD CL,BYTE PTR DS:[EAX]             |
00401102|.^EB F5        |JMP SHORT tto_Crac.004010F9          |
00401104|> 58           |POP EAX                              |
00401105|. 51           |PUSH ECX                             |
00401106|. 50           |PUSH EAX                             |
00401107|. E8 08060000  |CALL <JMP.&kernel32.IsDebuggerPresent>| IsDebuggerPresent
0040110C|. C3           |RETN                                 |
```

La première partie n'est pas très intéressante, elle ne fait que

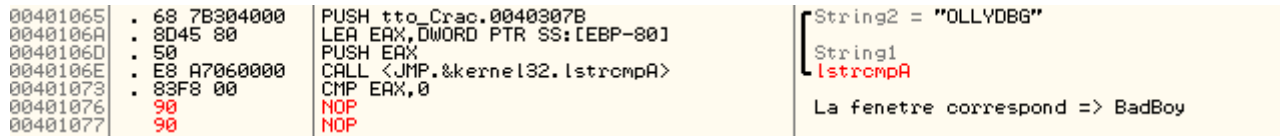
quelques verifications d'intégrité. Par contre il faut faire attention à l'appel à IsDebuggerPresent (fonction déterminant si un programme est débuggé ou non). En effet si l'on ne fait rien, cela pourrait nous emmener tout droit sur un BadBoy. Le plus simple est d'utiliser un plugin de Olly permettant de cacher notre debugger.

Viens ensuite l'appel à EnumWindows. Comme son nom l'indique cette fonction énumère toutes les fenêtres une par une et les fait traiter par la fonction passée en paramètre (ici 401048).

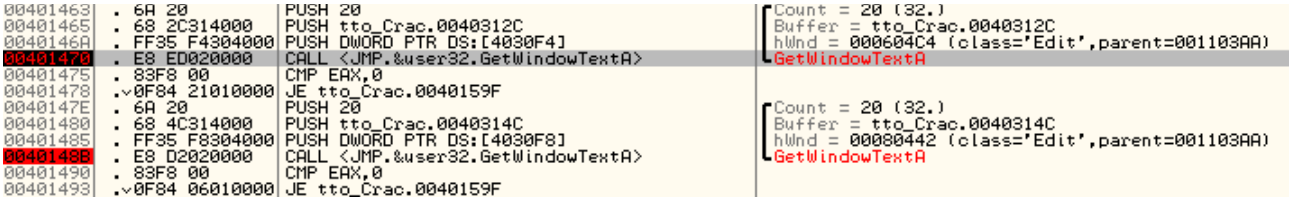
Ah bah ca tient mais alors ! On trouve des string qui correspondent comme par hasard à des debugger connus ! Il s'agit en fait des noms de classes de fenetres (fournies par GetClassNameA) de OllyDbg (OLLYDBG), WinDasm (OWL_Window) et IDA (TIdaWindow). Il y a donc des comparaisons pour verifier qu'aucun de ces debugger n'est actuellement lancé, sinon quoi on envoi un WM_CLOSE au debugger suivit d'un ExitProcess pour notre programme. La solution est toute simple : nopper le JE après la comparaison avec OLLYDBG (si bien sur vous utilisez Olly).

3.Trouver un triplet valide

Tous les anti-debug ayant été repérés, nous allons maintenant pouvoir passer en phase de debugging. Petite précaution toutefois : ne pas oublier de nopper en 401076 pour éviter que notre debugger se fasse fermer.

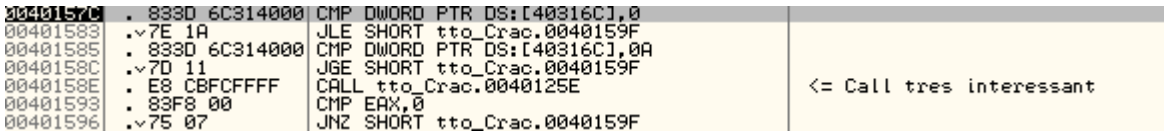


Il ne reste qu'à poser un BP sur les appels à GetWindowTextA (visibles par click droit -> Search For -> All intermodular Calls) et lancer le programme par F9.
Voilà le crackme se lance, on complète les infos (naspower/123456/987654 pour moi) on valide et on break ici :



En regardant de plus près on s'aperçoit que les 2 appels ne récupèrent ici que les serial. Qu'en est-il du name ?

On peut tracer sans problème jusqu'en 40157C (avant il n'y a que des controles d'intégrité sans intérêt).

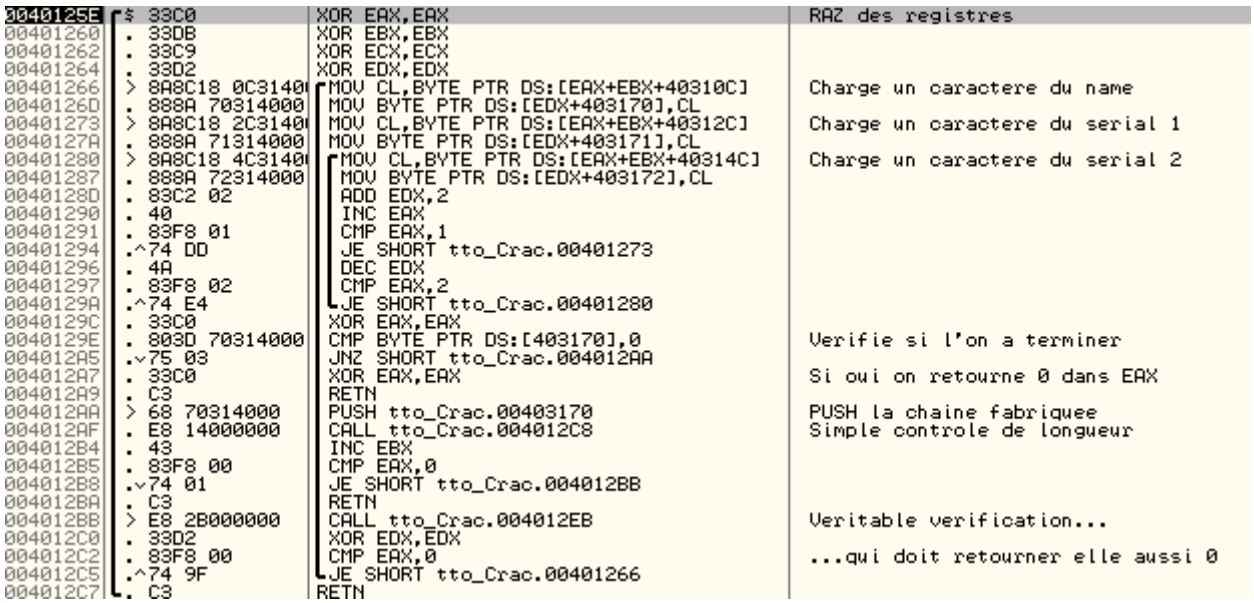


Si vous vous rappelez du début de ce tutorial vous saurez que 40316C contient le nombre de caractère du name. Ici il est donc clair que le serial doit obligatoirement faire strictement moins de A caractères (pour ceux dont ca ne serait pas le cas, il faut relancer sans oublier de nopper la ou il faut !). De plus en regardant notre name en 40310C on s'apercoit que toutes les lettres ont été passées en majuscules (notez ce détail ca servira plus tard).

Maintenant allons voir le Call que j'ai marqué comme intéressant.

En effet c'est lui qui détermine si les informations entrées sont correctes. Le petit CMP EAX, 0 juste après signifie qu'il faut que le Call renvoie une valeur nulle dans EAX.

40125E :



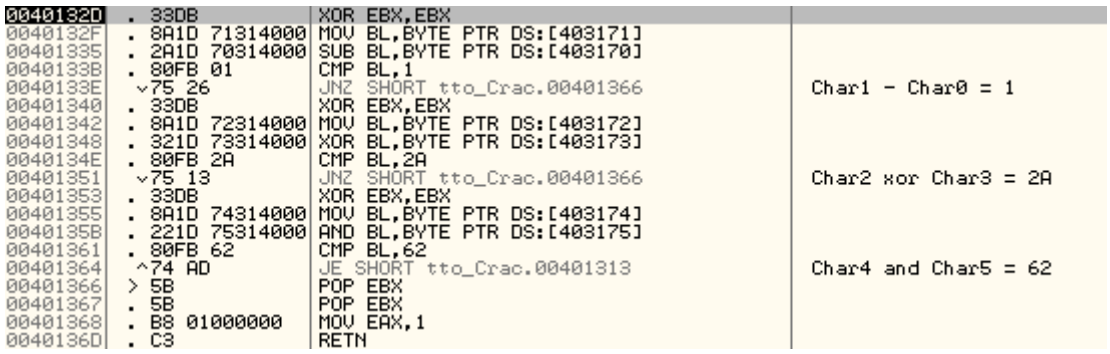
Dans cette routine on fabrique une chaine de 6 caractères composée dans l'ordre comme ceci :

- 1 caractère du name d'index i
- 1 caractère du serial 1 d'index i
- 1 caractère du serial 2 d'index i
- 1 caractère du serial 1 d'index i+1
- 1 caractère du serial 2 d'index i+1
- 1 caractère du serial 2 d'index i+2

Le Call vers 4012C8 n'est pas intéressant, il sert juste ici à vérifier que cette chaine fait bien 6 caractères de long.
Bien entendu cette boucle de vérification sera répétée tant qu'il y aura des caractères dans le name le serial 1 aura donc au moins 1 caractère de plus que le name et le serial 2 en aura au moins 2 de plus.

Intéressons nous maintenant au Call de 4012EB.

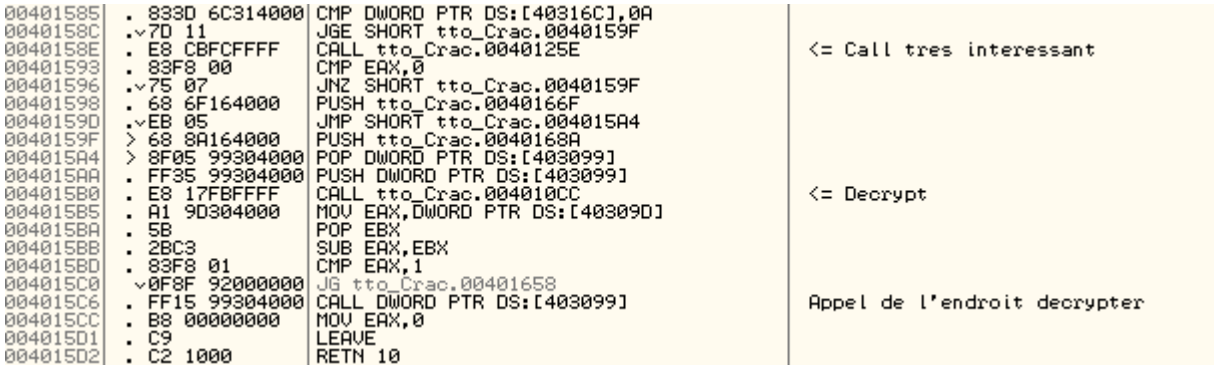
Encore une fois un peu d'instructions sans intérêt avant le vrai commencement en 40132D



Voilà nous avons enfin toutes les cartes dans nos mains !
Seulement ces 3 verifications devant être vérifiées pour toute chaine de caractères fabriquées, les choix vont être limités. Pour ma part j'ai pris la solution de facilité : je suppose que tous les caractères du serial 2 sont 62 ('b') (car 2 caractères successifs du serial 2 ANDé doivent donner 62). Je XOR 62 à 2A = 48 ('H') ce qui me donne tous les caractères du serial 1. Il ne reste qu'à soustraire 1 pour trouver les caractères du name qu'il faut repasser en minuscule.

On trouve donc le triplet gagnant (par exemple) : ggggg/HHHHHH/bbbbbbb

Mais tout est-il réellement fini ?
On clique un première fois rien ne se passe. Par contre on clique une deuxième fois et l'on a le droit au message « Bravo ». L'explication se trouve en 4015B0 peu après le Call intéressant de tout à l'heure :



Une fois que nos informations ont étées vérifiées et acceptée, un appel à une fonction de decryptage à lieu. Or l'endroit doit être décrypter 2 fois avant de révéler un appel à une MessageBox de félicitations :

```
0040166F  . C3              RETN                                    La premiere fois on tombe sur un RET
00401670  ? 59              POP ECX
00401671  . C159 89 99      RCR DWORD PTR DS:[ECX-77],99            Shift constant out of range 1..31
00401675  ? 59              POP ECX
00401676  ? C1C3 89         ROL EBX,89                              Shift constant out of range 1..31
00401679  . 99              CDQ
0040167A  . 59              POP ECX
0040167B  ? 58              POP EAX
0040167C  . 8E95 89995941   MOV SS,WORD PTR SS:[EBP+41599989]       Modification of segment register
00401682    41              DB 41                                   CHAR 'A'
00401683    59              DB 59                                   CHAR 'Y'
00401684    59              DB 59                                   CHAR 'Y'
00401685    59              DB 59                                   CHAR 'Y'
00401686    22              DB 22                                   CHAR '"'
00401687    1B              DB 1B
00401688    69              DB 69                                   CHAR 'i'
00401689  . 59              DB 59                                   CHAR 'Y'
0040168A  . C3              RETN
```

```
0040166F  . 6A 00           PUSH 0                                  La premiere fois on tombe sur un RET
00401671  . 68 00304000     PUSH tto_Crac.00403000                  ASCII "[lvl 2.5] CrackMe6 par Thierry The One"
00401676  ? 68 6A304000     PUSH tto_Crac.0040306A                  ASCII "Bravo"
0040167B  ? FF35 3C304000   PUSH DWORD PTR DS:[40303C]
00401681  ? E8 E8000000     CALL <JMP.&user32.MessageBoxA>
00401686    C9              DB C9
00401687    C2              DB C2
00401688    10              DB 10
00401689  . 00              DB 00
0040168A  . C3              RETN
```

Et voilà !

4.Conclusion

Ainsi s'achève l'étude de l'un des crackme que j'ai eu le plus de plaisir à étudier. En effet les différentes interconnections des parties du code apportent un petit plus à ce crackme.

Je tiens donc à féliciter Thierry The One pour ce crackme !
Et en espérant qu'il y en ai d'autres.

Encore une fois je tiens aussi à remercier Gamera pour son projet !

That's all folks !

26/09/06
mastermatt29^NAS